Anfis.py

# -\*- coding: utf-8 -\*-

"""

Created on Thu Apr 03 07:30:34 2014

@author: tim.meggs

"""

import itertools

import numpy as np

from anfis.membership import mfDerivs

import copy

class ANFIS:

    """

    Class to implement an Adaptive Network Fuzzy Inference System: ANFIS"

    Attributes:

        X

        Y

        XLen

        memClass

        memFuncs

        memFuncsByVariable

        rules

        consequents

        errors

        memFuncsHomo

        trainingType

    """

    def \_\_init\_\_(self, X, Y, memFunction):

        self.X = np.array(copy.copy(X))

        self.Y = np.array(copy.copy(Y))

        self.XLen = len(self.X)

        self.memClass = copy.deepcopy(memFunction)

        self.memFuncs = self.memClass.MFList

        self.memFuncsByVariable = [[x for x in range(len(self.memFuncs[z]))] for z in range(len(self.memFuncs))]

        self.rules = np.array(list(itertools.product(\*self.memFuncsByVariable)))

        self.consequents = np.empty(self.Y.ndim \* len(self.rules) \* (self.X.shape[1] + 1))

        self.consequents.fill(0)

        self.errors = np.empty(0)

        self.memFuncsHomo = all(len(i)==len(self.memFuncsByVariable[0]) for i in self.memFuncsByVariable)

        self.trainingType = 'Not trained yet'

    def LSE(self, A, B, initialGamma = 1000.):

        coeffMat = A

        rhsMat = B

        S = np.eye(coeffMat.shape[1])\*initialGamma

        x = np.zeros((coeffMat.shape[1],1)) # need to correct for multi-dim B

        for i in range(len(coeffMat[:,0])):

            a = coeffMat[i,:]

            b = np.array(rhsMat[i])

            S = S - (np.array(np.dot(np.dot(np.dot(S,np.matrix(a).transpose()),np.matrix(a)),S)))/(1+(np.dot(np.dot(S,a),a)))

            x = x + (np.dot(S,np.dot(np.matrix(a).transpose(),(np.matrix(b)-np.dot(np.matrix(a),x)))))

        return x

    def trainHybridJangOffLine(self, epochs=5, tolerance=1e-5, initialGamma=1000, k=0.01):

        self.trainingType = 'trainHybridJangOffLine'

        convergence = False

        epoch = 1

        while (epoch < epochs) and (convergence is not True):

            #layer four: forward pass

            [layerFour, wSum, w] = forwardHalfPass(self, self.  X)

            #layer five: least squares estimate

            layerFive = np.array(self.LSE(layerFour,self.Y,initialGamma))

            self.consequents = layerFive

            layerFive = np.dot(layerFour,layerFive)

            #error

            error = np.sum((self.Y-layerFive.T)\*\*2)

            print('current error: '+ str(error))

            average\_error = np.average(np.absolute(self.Y-layerFive.T))

            self.errors = np.append(self.errors,error)

            if len(self.errors) != 0:

                if self.errors[len(self.errors)-1] < tolerance:

                    convergence = True

            # back propagation

            if convergence is not True:

                cols = range(len(self.X[0,:]))

                dE\_dAlpha = list(backprop(self, colX, cols, wSum, w, layerFive) for colX in range(self.X.shape[1]))

            if len(self.errors) >= 4:

                if (self.errors[-4] > self.errors[-3] > self.errors[-2] > self.errors[-1]):

                    k = k \* 1.1

            if len(self.errors) >= 5:

                if (self.errors[-1] < self.errors[-2]) and (self.errors[-3] < self.errors[-2]) and (self.errors[-3] < self.errors[-4]) and (self.errors[-5] > self.errors[-4]):

                    k = k \* 0.9

            ## handling of variables with a different number of MFs

            t = []

            for x in range(len(dE\_dAlpha)):

                for y in range(len(dE\_dAlpha[x])):

                    for z in range(len(dE\_dAlpha[x][y])):

                        t.append(dE\_dAlpha[x][y][z])

            eta = k / np.abs(np.sum(t))

            if(np.isinf(eta)):

                eta = k

            ## handling of variables with a different number of MFs

            dAlpha = copy.deepcopy(dE\_dAlpha)

            if not(self.memFuncsHomo):

                for x in range(len(dE\_dAlpha)):

                    for y in range(len(dE\_dAlpha[x])):

                        for z in range(len(dE\_dAlpha[x][y])):

                            dAlpha[x][y][z] = -eta \* dE\_dAlpha[x][y][z]

            else:

                dAlpha = -eta \* np.array(dE\_dAlpha)

            for varsWithMemFuncs in range(len(self.memFuncs)):

                for MFs in range(len(self.memFuncsByVariable[varsWithMemFuncs])):

                    paramList = sorted(self.memFuncs[varsWithMemFuncs][MFs][1])

                    for param in range(len(paramList)):

                        self.memFuncs[varsWithMemFuncs][MFs][1][paramList[param]] = self.memFuncs[varsWithMemFuncs][MFs][1][paramList[param]] + dAlpha[varsWithMemFuncs][MFs][param]

            epoch = epoch + 1

        self.fittedValues = predict(self,self.X)

        self.residuals = self.Y - self.fittedValues[:,0]

        return self.fittedValues

    def plotErrors(self):

        if self.trainingType == 'Not trained yet':

            print(self.trainingType)

        else:

            import matplotlib.pyplot as plt

            plt.plot(range(len(self.errors)),self.errors,'ro', label='errors')

            plt.ylabel('error')

            plt.xlabel('epoch')

            plt.show()

    def plotMF(self, x, inputVar):

        import matplotlib.pyplot as plt

        from skfuzzy import gaussmf, gbellmf, sigmf

        for mf in range(len(self.memFuncs[inputVar])):

            if self.memFuncs[inputVar][mf][0] == 'gaussmf':

                y = gaussmf(x,\*\*self.memClass.MFList[inputVar][mf][1])

            elif self.memFuncs[inputVar][mf][0] == 'gbellmf':

                y = gbellmf(x,\*\*self.memClass.MFList[inputVar][mf][1])

            elif self.memFuncs[inputVar][mf][0] == 'sigmf':

                y = sigmf(x,\*\*self.memClass.MFList[inputVar][mf][1])

            plt.plot(x,y,'r')

        plt.show()

    def plotResults(self):

        if self.trainingType == 'Not trained yet':

            print(self.trainingType)

        else:

            import matplotlib.pyplot as plt

            plt.plot(range(len(self.fittedValues)),self.fittedValues,'r', label='trained')

            plt.plot(range(len(self.Y)),self.Y,'b', label='original')

            plt.legend(loc='upper left')

            plt.show()

def forwardHalfPass(ANFISObj, Xs):

    layerFour = np.empty(0,)

    wSum = []

    for pattern in range(len(Xs[:,0])):

        #layer one

        layerOne = ANFISObj.memClass.evaluateMF(Xs[pattern,:])

        #layer two

        miAlloc = [[layerOne[x][ANFISObj.rules[row][x]] for x in range(len(ANFISObj.rules[0]))] for row in range(len(ANFISObj.rules))]

        layerTwo = np.array([np.product(x) for x in miAlloc]).T

        if pattern == 0:

            w = layerTwo

        else:

            w = np.vstack((w,layerTwo))

        #layer three

        wSum.append(np.sum(layerTwo))

        if pattern == 0:

            wNormalized = layerTwo/wSum[pattern]

        else:

            wNormalized = np.vstack((wNormalized,layerTwo/wSum[pattern]))

        #prep for layer four (bit of a hack)

        layerThree = layerTwo/wSum[pattern]

        rowHolder = np.concatenate([x\*np.append(Xs[pattern,:],1) for x in layerThree])

        layerFour = np.append(layerFour,rowHolder)

    w = w.T

    wNormalized = wNormalized.T

    layerFour = np.array(np.array\_split(layerFour,pattern + 1))

    return layerFour, wSum, w

def backprop(ANFISObj, columnX, columns, theWSum, theW, theLayerFive):

    paramGrp = [0]\* len(ANFISObj.memFuncs[columnX])

    for MF in range(len(ANFISObj.memFuncs[columnX])):

        parameters = np.empty(len(ANFISObj.memFuncs[columnX][MF][1]))

        timesThru = 0

        for alpha in sorted(ANFISObj.memFuncs[columnX][MF][1].keys()):

            bucket3 = np.empty(len(ANFISObj.X))

            for rowX in range(len(ANFISObj.X)):

                varToTest = ANFISObj.X[rowX,columnX]

                tmpRow = np.empty(len(ANFISObj.memFuncs))

                tmpRow.fill(varToTest)

                bucket2 = np.empty(ANFISObj.Y.ndim)

                for colY in range(ANFISObj.Y.ndim):

                    rulesWithAlpha = np.array(np.where(ANFISObj.rules[:,columnX]==MF))[0]

                    adjCols = np.delete(columns,columnX)

                    senSit = mfDerivs.partial\_dMF(ANFISObj.X[rowX,columnX],ANFISObj.memFuncs[columnX][MF],alpha)

                    # produces d\_ruleOutput/d\_parameterWithinMF

                    dW\_dAplha = senSit \* np.array([np.prod([ANFISObj.memClass.evaluateMF(tmpRow)[c][ANFISObj.rules[r][c]] for c in adjCols]) for r in rulesWithAlpha])

                    bucket1 = np.empty(len(ANFISObj.rules[:,0]))

                    for consequent in range(len(ANFISObj.rules[:,0])):

                        fConsequent = np.dot(np.append(ANFISObj.X[rowX,:],1.),ANFISObj.consequents[((ANFISObj.X.shape[1] + 1) \* consequent):(((ANFISObj.X.shape[1] + 1) \* consequent) + (ANFISObj.X.shape[1] + 1)),colY])

                        acum = 0

                        if consequent in rulesWithAlpha:

                            acum = dW\_dAplha[np.where(rulesWithAlpha==consequent)] \* theWSum[rowX]

                        acum = acum - theW[consequent,rowX] \* np.sum(dW\_dAplha)

                        acum = acum / theWSum[rowX]\*\*2

                        bucket1[consequent] = fConsequent \* acum

                    sum1 = np.sum(bucket1)

                    if ANFISObj.Y.ndim == 1:

                        bucket2[colY] = sum1 \* (ANFISObj.Y[rowX]-theLayerFive[rowX,colY])\*(-2)

                    else:

                        bucket2[colY] = sum1 \* (ANFISObj.Y[rowX,colY]-theLayerFive[rowX,colY])\*(-2)

                sum2 = np.sum(bucket2)

                bucket3[rowX] = sum2

            sum3 = np.sum(bucket3)

            parameters[timesThru] = sum3

            timesThru = timesThru + 1

        paramGrp[MF] = parameters

    return paramGrp

def predict(ANFISObj, varsToTest):

    [layerFour, wSum, w] = forwardHalfPass(ANFISObj, varsToTest)

    #layer five

    layerFive = np.dot(layerFour,ANFISObj.consequents)

    return layerFive

if \_\_name\_\_ == "\_\_main\_\_":

    print("I am main!")

Membershipfunction.py

# -\*- coding: utf-8 -\*-

"""

Created on Mon Mar 31 15:41:58 2014

@author: tim.meggs

"""

from skfuzzy import gaussmf, gbellmf, sigmf

class MemFuncs:

    'Common base class for all employees'

    funcDict = {'gaussmf': gaussmf, 'gbellmf': gbellmf, 'sigmf': sigmf}

    def \_\_init\_\_(self, MFList):

        self.MFList = MFList

    def evaluateMF(self, rowInput):

        if len(rowInput) != len(self.MFList):

            print("Number of variables does not match number of rule sets")

        return [[self.funcDict[self.MFList[i][k][0]](rowInput[i],\*\*self.MFList[i][k][1]) for k in range(len(self.MFList[i]))] for i in range(len(rowInput))]

mfDerives.py

    import numpy as np

    def partial\_dMF(x, mf\_definition, partial\_parameter):

        """Calculates the partial derivative of a membership function at a point x.

        Parameters

        ------

        Returns

        ------

        """

        mf\_name = mf\_definition[0]

        if mf\_name == 'gaussmf':

            sigma = mf\_definition[1]['sigma']

            mean = mf\_definition[1]['mean']

            if partial\_parameter == 'sigma':

                result = (2./sigma\*\*3) \* np.exp(-(((x-mean)\*\*2)/(sigma)\*\*2))\*(x-mean)\*\*2

            elif partial\_parameter == 'mean':

                result = (2./sigma\*\*2) \* np.exp(-(((x-mean)\*\*2)/(sigma)\*\*2))\*(x-mean)

        elif mf\_name == 'gbellmf':

            a = mf\_definition[1]['a']

            b = mf\_definition[1]['b']

            c = mf\_definition[1]['c']

            if partial\_parameter == 'a':

                result = (2. \* b \* np.power((c-x),2) \* np.power(np.absolute((c-x)/a), ((2 \* b) - 2))) / \

                    (np.power(a, 3) \* np.power((np.power(np.absolute((c-x)/a),(2\*b)) + 1), 2))

            elif partial\_parameter == 'b':

                result = -1 \* (2 \* np.power(np.absolute((c-x)/a), (2 \* b)) \* np.log(np.absolute((c-x)/a))) / \

                    (np.power((np.power(np.absolute((c-x)/a), (2 \* b)) + 1), 2))

            elif partial\_parameter == 'c':

                result = (2. \* b \* (c-x) \* np.power(np.absolute((c-x)/a), ((2 \* b) - 2))) / \

                    (np.power(a, 2) \* np.power((np.power(np.absolute((c-x)/a),(2\*b)) + 1), 2))

        elif mf\_name == 'sigmf':

            b = mf\_definition[1]['b']

            c = mf\_definition[1]['c']

            if partial\_parameter == 'b':

                result = -1 \* (c \* np.exp(c \* (b + x))) / \

                    np.power((np.exp(b\*c) + np.exp(c\*x)), 2)

            elif partial\_parameter == 'c':

                result = ((x - b) \* np.exp(c \* (x - b))) / \

                    np.power((np.exp(c \* (x - c))) + 1, 2)

        return result

tests.py

from anfis.anfis import ANFIS

import membership.mfDerivs

import membership.membershipfunction

import numpy

ts = numpy.loadtxt("trainingSet.txt", usecols=[1,2,3])#numpy.loadtxt('c:\\Python\_fiddling\\myProject\\MF\\trainingSet.txt',usecols=[1,2,3])

X = ts[:,0:2]

Y = ts[:,2]

mf = [

       [

           ['gaussmf',{'mean':0.,'sigma':1.}],

           ['gaussmf',{'mean':-1.,'sigma':2.}],

           ['gaussmf',{'mean':-4.,'sigma':10.}],

           ['gaussmf',{'mean':-7.,'sigma':7.}]

        ],

       [

           ['gaussmf',{'mean':1.,'sigma':2.}],

           ['gaussmf',{'mean':2.,'sigma':3.}],

           ['gaussmf',{'mean':-2.,'sigma':10.}],

           ['gaussmf',{'mean':-10.5,'sigma':5.}]

        ]

     ]

mfc = membership.membershipfunction.MemFuncs(mf)

anf = ANFIS(X, Y, mfc)

anf.trainHybridJangOffLine(epochs=200)

print(round(anf.consequents[-1][0],6))

print(round(anf.consequents[-2][0],6))

print(round(anf.fittedValues[9][0],6))

if round(anf.consequents[-1][0],6) == -5.275538 and round(anf.consequents[-2][0],6) == -1.990703 and round(anf.fittedValues[9][0],6) == 0.002249:

    print('test is good')

anf.plotErrors()

anf.plotResults()

TrainingSet.txt

"1" 25.331 2.859 28.646

"2" 25.283 2.855 29.438

"3" 25.247 2.855 29.438

"4" 25.195 2.813 29.438

"5" 25.151 2.809 30.317

"6" 25.111 2.808 30.317

"7" 25.068 2.76 30.317

"8" 25.038 2.757 31.221

"9" 24.991 2.741 31.221

"10" 24.947 2.704 31.221

"11" 24.907 2.703 32.142

"12" 24.863 2.672 32.142

"13" 24.834 2.65 32.142

"15" 24.789 2.649 33.038

"16" 24.752 2.612 33.038

"17" 24.712 2.596 33.038

"18" 24.666 2.596 33.971

"19" 24.641 2.559 33.971

"20" 24.594 2.551 33.971

"" 24.565 2.551 34.713

"" 24.512 2.53 34.713

"" 24.465 2.528 34.713

"" 24.435 2.528 34.992

"" 24.388 2.531 34.992

"" 24.352 2.532 34.992

"" 24.297 2.536 34.83

"" 24.247 2.561 34.83

"" 24.211 2.562 34.83

"" 24.164 2.579 34.255

"" 24.13 2.601 34.255

"" 24.068 2.602 34.255

"" 24.018 2.626 33.809

"" 23.978 2.64 33.809

"" 23.928 2.64 33.809

"" 23.892 2.665 33.496

"" 23.839 2.673 33.496

"" 23.785 2.673 33.496

"" 23.747 2.695 33.3

"" 23.695 2.699 33.3

"" 23.66 2.699 33.3

"" 23.602 2.716 33.176

"" 23.552 2.717 33.176

"" 23.513 2.718 33.176

"" 23.465 2.731 33.13

"" 23.431 2.731 33.13

"" 23.377 2.734 33.13

"" 23.327 2.74 33.059

"" 23.284 2.74 33.059

"" 23.237 2.743 33.059

"" 23.207 2.745 33.155

"" 23.155 2.745 33.155

"" 23.105 2.746 33.155

"" 23.065 2.746 33.271

"" 23.019 2.746 33.271

"" 22.987 2.746 33.271

"" 22.94 2.746 33.417

"" 22.894 2.746 33.417

"" 22.854 2.745 33.417

"" 22.804 2.745 33.555

"" 22.775 2.745 33.555

"" 22.729 2.739 33.555

"" 22.688 2.739 33.784

"" 22.643 2.737 33.784

"" 22.596 2.732 33.784

"" 22.57 2.732 34.009

"" 22.528 2.727 34.009

"" 22.489 2.723 34.009

"" 22.441 2.723 34.271

"" 22.396 2.715 34.271

"" 22.36 2.712 34.271

"" 22.319 2.712 34.559

"" 22.292 2.702 34.559

"" 22.24 2.7 34.559

"" 22.195 2.7 34.888

"" 22.163 2.687 34.888

"" 22.124 2.686 34.888

"" 22.092 2.686 35.251

"" 22.043 2.671 35.251

"" 21.997 2.67 35.251

"" 21.966 2.668 35.638

"" 21.931 2.653 35.638

"" 21.902 2.653 35.638

"" 21.852 2.646 36.072

"" 21.809 2.634 36.072

"" 21.775 2.634 36.072

"" 21.775 2.634 36.072

"" 21.737 2.623 36.551

"" 21.709 2.617 36.551

"" 21.666 2.617 36.551

"" 21.619 2.609 36.963

"" 21.587 2.607 36.963

"" 21.546 2.607 36.963

"" 21.522 2.608 37.184

"" 21.472 2.608 37.184

"" 21.426 2.608 37.184

"" 21.392 2.62 37.242

"" 21.351 2.621 37.242

"" 21.322 2.621 37.242

"" 21.274 2.636 37.076

"" 21.229 2.636 37.076

"" 21.193 2.642 37.076

"" 21.152 2.652 36.876

"" 21.123 2.652 36.876

"" 21.075 2.659 36.876

"" 21.036 2.664 36.851

"" 20.994 2.664 36.851

"" 20.951 2.671 36.851

"" 20.923 2.673 36.93

"" 20.88 2.673 36.93

"" 20.844 2.679 36.93

"" 20.803 2.68 37.022

"" 20.756 2.68 37.022

"" 20.731 2.684 37.022

"" 20.688 2.684 37.172

"" 20.653 2.684 37.172

"" 20.606 2.684 37.172

"" 20.563 2.684 37.368

"" 20.536 2.684 37.368

"" 20.495 2.684 37.368

"" 20.468 2.684 37.522

"" 20.422 2.682 37.522

"" 20.377 2.68 37.522

"" 20.348 2.68 37.792

"" 20.309 2.676 37.792

"" 20.284 2.674 37.792

"" 20.239 2.674 38.042

"" 20.191 2.668 38.042

"" 20.162 2.667 38.042

"" 20.126 2.667 38.334

"" 20.099 2.659 38.334

"" 20.055 2.658 38.334

"" 20.014 2.658 38.638

"" 19.987 2.649 38.638

"" 19.949 2.649 38.638

"" 19.924 2.649 38.934

"" 19.879 2.64 38.934

"" 19.835 2.64 38.934

"" 19.806 2.636 39.264

"" 19.77 2.63 39.264

"" 19.747 2.629 39.264

"" 19.704 2.623 39.609

"" 19.661 2.619 39.609

"" 19.632 2.618 39.609

"" 19.596 2.613 39.972

"" 19.571 2.611 39.972

"" 19.532 2.611 39.972

"" 19.493 2.61 40.239

"" 19.457 2.609 40.239

"" 19.419 2.609 40.239

"" 19.394 2.614 40.376

"" 19.353 2.615 40.376

"" 19.314 2.615 40.376

"" 19.276 2.629 40.409

"" 19.233 2.63 40.409

"" 19.212 2.634 40.409

"" 19.169 2.645 40.134

"" 19.135 2.646 40.134

"" 19.095 2.653 40.134

"" 19.052 2.659 40.059

"" 19.033 2.659 40.059

"" 18.988 2.666 40.059

"" 18.961 2.67 40.118

"" 18.916 2.67 40.118

"" 18.873 2.676 40.118

"" 18.852 2.677 40.184

"" 18.809 2.678 40.184

"" 18.786 2.682 40.184

"" 18.739 2.682 40.326

"" 18.702 2.682 40.326

"" 18.671 2.684 40.326

"" 18.634 2.684 40.489

"" 18.603 2.684 40.489

"" 18.562 2.684 40.489

"" 18.524 2.684 40.668

"" 18.494 2.684 40.668

"" 18.463 2.684 40.668

"" 18.437 2.684 40.814

"" 18.394 2.681 40.814

"" 18.351 2.68 40.814

"" 18.324 2.68 41.101

"" 18.29 2.675 41.101

"" 18.267 2.674 41.101

"" 18.229 2.674 41.36

"" 18.188 2.667 41.36

"" 18.161 2.666 41.36

"" 18.123 2.666 41.651

"" 18.102 2.657 41.651

"" 18.063 2.657 41.651

"" 18.023 2.656 41.964

"" 17.998 2.647 41.964

"" 17.962 2.647 41.964

"" 17.943 2.644 42.293

"" 17.901 2.637 42.293

"" 17.866 2.637 42.293

"" 17.835 2.631 42.605

"" 17.796 2.627 42.605

"" 17.78 2.627 42.605

"" 17.742 2.619 42.955

"" 17.708 2.616 42.955

"" 17.678 2.615 42.955

"" 17.637 2.608 43.33

"" 17.624 2.607 43.33

"" 17.581 2.607 43.33

"" 17.549 2.605 43.601

"" 17.513 2.605 43.601

"" 17.475 2.605 43.601

"" 17.459 2.605 43.751

"" 17.42 2.605 43.751

"" 17.393 2.606 43.751

"" 17.354 2.609 43.926

"" 17.313 2.609 43.926

"" 17.291 2.611 43.926

"" 17.257 2.613 43.993

"" 17.236 2.613 43.993

"" 17.191 2.616 43.993

"" 17.155 2.617 44.151

"" 17.13 2.617 44.151

"" 17.1 2.618 44.151

"" 17.078 2.618 44.364

"" 17.033 2.618 44.364

"" 16.997 2.618 44.364

"" 16.976 2.618 44.593

"" 16.944 2.618 44.593

"" 16.919 2.618 44.593

"" 16.878 2.618 44.801

"" 16.84 2.618 44.801

"" 16.811 2.616 44.801

"" 16.786 2.616 45.06

"" 16.765 2.616 45.06

"" 16.722 2.616 45.06

"" 16.682 2.616 45.239

"" 16.657 2.621 45.239

